**2Б**

**УДК 681.3**

***М.К. Буза***
Белорусский государственный университет, г. Минск, Республика Беларусь
bouza@bsu.by

# Параллельные вычисления на графических процессорах

Рассматривается один из подходов сокращения времени вычислений хорошо распараллеливаемых алгоритмов, основанный на совместном использовании классических и графических процессоров. Проводится анализ эффективности такой организации вычислений и выясняются «узкие» места. На конкретных примерах показываются возможности подобного тандема и формулируются требования к алгоритмам для их эффективного исполнения.

## Введение

Практически любая современная платформа исполнения программного кода, будь то полноценная операционная система или виртуальная машина (например, JAVA машина или .NET framework), поддерживающая мультизадачность, содержит набор API, предназначенный для управления потоками и создания параллельных программ. Таким образом, имеется возможность организовать параллельные вычисления практически на любом языке от Assembler до скриптовых языков типа Perl. Ясно, что проектировать параллельные программы не всегда оправданное решение с точки зрения временных затрат и качества кода, так как на разработчика часто ложится множество специфических рутинных задач по созданию, управлению, контролю и обеспечению синхронизации потоков выполнения. Речь идет, безусловно, о решении вычислительно трудных задач, так как прикладные программы создаются главным образом, используя API платформы. На сегодняшний день имеются библиотеки и языки параллельного программирования, что упрощает множество проблем, предоставляя пользователю механизмы для организации параллельных вычислений. Среди них можно отметить MPI, PVM, языки Cisal, NESL, ZPL, Java, а также расширение всевозможных языков программирования.

В последнее время начали уделять внимание концепции GPGPU (General-purpose graphics processing units) – технологии использования графического процессора видеокарты для общих вычислений, которые обычно выполняет центральный процессор.

В связи с этим была поставлена следующая **задача** – исследовать возможности использования видеокарт для общих вычислений и оценить их эффективность.

**Цель работы** – повышение скорости обработки хорошо распараллеливаемых алгоритмов на тандеме CPU + GPU и выяснение «узких» мест в этой технологии.

## Графические процессоры

Универсальные устройства с многоядерными процессорами для параллельных векторных вычислений, используемых в 3D-графике, достигают высокой пиковой производительности, которой не могут достичь универсальные процессоры. Конечно, максимальная скорость достигается на задачах определенного класса.

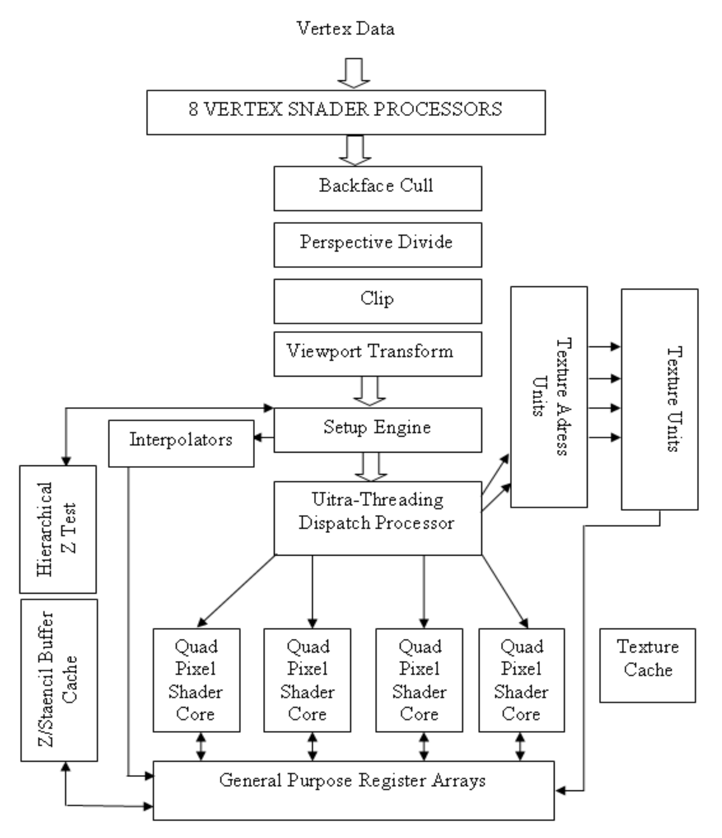



Рисунок 1 – Схема графического процессора ATI Radeon X1800

Примером такого параллельного процессора может служить процессор Cell, разработанный альянсом Sony-Toshiba-IBM и применяемый в игровой приставке Sony PlayStation 3, а также современные видеокарты от лидеров рынка – компаний Nvidia и AMD.

Так, например, процесс формирования изображения в графическом процессоре ATI Radeon X1800 (рис. 1) выполняется следующим образом. Вначале данные (Vertex Data) обрабатываются восемью вершинными процессорами для вычисления геометрии трехмерного изображения. Затем блок сборки (setup engine) осуществляет растризацию геометрии, после чего идет процесс организации многопоточной обработки (блок Ultra-Threading Dispatch Processor), т.е. распараллеливание шейдерного кода на многие сотни потоков. И, наконец, выполняют свои функции пиксельные процессоры.

# Необходимость перехода к процессорам общего назначения

Анализ различных графических задач показал, что строгое закрепление функций за вершинными и пиксельными процессорами не эффективно. Действительно, если трехмерные модели имеют насыщенную геометрию, то в основном используются вершинные процессоры, а если модель насыщена пиксельными эффектами, то основная работа отводится пиксельным процессорам. Это, в основном, и послужило аргументацией для компании Nvidia для перехода на процессоры общего назначения. Последние способны выполнять не только функции вершинных и пиксельных процессоров, но и выполнять различные расчетные работы общего назначения.

Анализ шейдерных программ, применяемых Nvidia и ATI, показал неэффективность использования вычислительных ресурсов при векторной архитектуре исполнительных блоков. Это привело к пониманию необходимости перехода в унифицированных про-

цессорах к скалярным вычислениям, поручив работу по преобразованию векторных операций в скалярные самому GPU, что и было сделано компанией Nvidia в графическом процессоре GeForce 8800. Этот процессор содержит 128 потоковых унифицированных процессоров, каждый из которых работает с частотой 1,35 ГГц.

Согласно архитектуре GeForce 8800 входные данные (input stream) процессором обрабатываются, а его выход (output stream) идет на вход другого процессора для последующей обработки. Циклическая потоковая обработка позволяет, если необходимо, произвести повторную обработку данных, что встречается довольно часто в графических построениях, без повторного ввода исходных данных либо для их последующих преобразований.

Для 3D видеоускорителей несколько лет назад появились первые технологии неграфических расчетов. Современные видеочипы содержат сотни математических исполнительных блоков, и эта мощь может использоваться для значительного ускорения множества вычислительно интенсивных приложений. Нынешние поколения GPU обладают достаточно гибкой архитектурой, что вместе с высокоуровневыми языками программирования делает их значительно более доступными для сложных вычислений.

Использование суперкомпьютеров для параллельных вычислений становится все более дорогим удовольствием. Кроме того, суперкомпьютеры требуют постоянной модернизации для поддержки эффективности вычислений. В связи с этим происходит постепенный перевод научных вычислений на другие платформы. Применение GPU процессоров позволяет повысить эффективность вычислений, используя специализированные библиотеки. Видеокарты достаточно дешевы, легко заменяемы и до определенных пределов их количество можно наращивать без особых трудностей. Так, например, специализированный графический процессор – TESLA C2070 (Nvidia) имеет один GPU процессор, состоящий из 448 вычислительных ядер, работающих параллельно с пиковой производительностью 515 Gflops, выполняя операции с плавающей точкой двойной точности или 1,03 Tflops с одинарной точностью, снабжен 6GB высокоскоростной видеопамяти GDDR5, работающей на частоте 1,5 GHz, при цене 3999$.

Сегодня наиболее часто используются четыре платформы, реально воплотившие концепцию GPGPU.

AMD FireStream – технология GPGPU, позволяющая программистам реализовывать алгоритмы, выполняемые на графических процессорах ускорителей ATI.

CUDA – технология GPGPU, позволяющая реализовывать на языке программирования C алгоритмы, выполняемые на графических процессорах ускорителей GeForce восьмого поколения и старше (GeForce 8 Series, GeForce 9 Series, GeForce 200, 300, 400 Series), Nvidia Quarod и Nvidia Tesla компании Nvidia. Технология CUDA разработана компанией Nvidia. CUDA ToolKit 3.0 (Nvidia) поддерживает OpenCL.

DirectCompute – набор интерфейсов программирования приложений (API) компании Microsoft является частью последних версий DirectX. Он предназначен для выполнения вычислений общего назначения на графических процессорах. Безусловно, он может использоваться и в игровой практике. Поддерживается компаниями AMD и Nvidia.

OpenCL является языком программирования задач, связанных с параллельными вычислениями на различных графических и центральных процессорах. Язык программирования базируется на стандарте C99. OpenCL включает также интерфейс программирования приложений и обеспечивает параллелизм на уровне инструкций и на уровне данных.

DirectCompute – интерфейс программирования приложений (API), который входит в состав DirectX – набора API от Microsoft, который предназначен для работы на IBM PC-совместимых компьютерах под управлением операционных систем семейства Microsoft Windows.

DirectCompute, появившись в составе DirectX 11, по существу стал первой технологией в составе DirectX, предоставившей доступ к вычислениям общего назначения на графических процессорах.

Несмотря на ориентацию на неграфические вычисления общего назначения, DirectCompute может использоваться и в игровой графике, например, при рендеринге теней, рендеринге полупрозрачных поверхностей без предварительной сортировки, а также при обработке и фильтрации цифровых изображений, просчете алгоритмов игрового искусственного интеллекта и для других задач.

Следует отметить, что множество приложений по молекулярному моделированию хорошо приспособлено для расчетов на видеочипах.

На сегодняшний день GPU предлагают высокую производительность. Так, например, GPU (Evergreen) позиционируют производительность до $10^{12}$ оп/с. Поэтому разумное использование тандема (CPU, GPU) позволит существенно ускорить обработку больших массивов данных.

Основные различия между GPU и CPU состоят в архитектурных решениях (потоковая и топографическая) и организации памяти (иерархическая и модель с максимальной пропускной способностью).

Однако широко используемые сегодня модели программирования GPU (CUDA и OpenCL) являются низкоуровневыми, что требует от программиста значительных усилий по проектированию практически полного сценария обработки данных, в том числе в трансформации обычных программ и управления ресурсами.

Типичная программа вычислений на CUDA выполняет следующие действия:

– копирует необходимые данные из оперативной памяти CPU в оперативную память GPU;

– задает размерность блоков вычислений, их количество и инициализирует процесс вычислений на GUDA;

– каждый вновь появившийся поток копирует часть необходимых для выполнения блока данных в быструю разделяемую память;

– выполняет вычисления;

– копирует результаты выполнения в основную память GPU;

– копирует результаты из памяти GPU в основную оперативную память компьютера.

# Технология общих вычислений

В вычислительной модели CUDA графический процессор GPU можно рассматривать как сопроцессор к CPU с собственной памятью, на который передаются вычисления для параллельной обработки большого числа потоков. При этом накладные расходы на управление параллельной обработкой на GPU минимальны по сравнению с аналогичной акцией на CPU. Ясно, что чем больше потоков будет обрабатываться на графических процессорах, тем эффективнее они будут использоваться.

Обработка на GPU + CPU осуществляется по следующей схеме (рис. 2).

При этом CUDA предоставляет разработчику эффективных программистских проектов ряд функций, которые могут исполняться только на CPU, так называемый CUDA host API.

Система CUDA обладает возможностью автоматического разбиения обрабатываемой части на потоки и управление ими. При этом все потоки организовываются в иерархию: все множество потоков (grid), блоки и отдельные потоки. Ясно, что взаимодействие между потоками лучше не допускать, чтобы стоимость обработки была минимальной. Но если такое взаимодействие необходимо, то в CUDA для этого есть два механизма: разделяемая (shared) память; барьерная синхронизация.

Так как обработка потоков идет по технологии SIMD, то фактически выполняется одна инструкция, но над различными данными. При этом исходная программа разрабатывается на «урезанном» C (нет операций ввода/вывода, ряд функций не поддерживаются и т. д.), а соответствующие файлы имеют расширение *.cu*. Если некоторые функции могут выполняться как на CPU, так и на GPU, то соответствующие спецификаторы *host* и *device* могут использоваться вместе.

Компилятор автоматически сгенерирует код для обеих платформ.

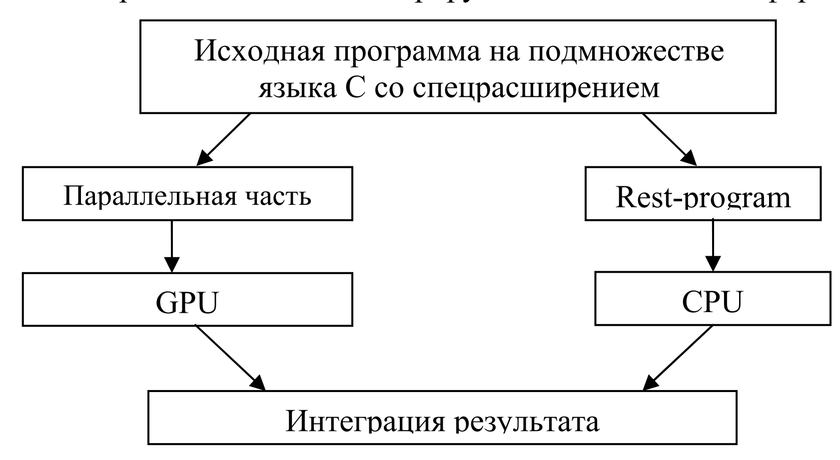



Рисунок 2 – Структура обработки программ на тандеме GPU + CPU

Для каждого потока будут известны: индекс потока внутри блока (*threadldx*, по умолчанию они предполагаются трехмерными) и индекс блока внутри сетей (*blockldx*).

Блок потоков выполняется на мультипроцессоре пулами, каждый из которых включает, как правило, 32 потока. При этом внутри пула одновременно может выполняться только одна инструкция. Пул разбивается на части, кратные количеству процессоров в мультипроцессоре и выполняются последовательно. Обмен данными между задачами внутри блока осуществляется через общую разделяемую память.

Безусловно, возможны различные эвристики по взаимодействию процессов, но они учитывают особенности реальной аппаратуры и собственные системные наработки. В качестве последних можно использовать, например, среду реализации Common Language Runtime (CLR) платформы .NET. Это позволяет сделать реализацию преобразования алгоритмов решения задач на GPU независимой от платформы и языка программирования.

Чтобы выполнить параллельные фрагменты на GPU необходимо:

– загрузить модуль с необходимыми функциями для видеокарты;

– выделить необходимый объем памяти на GPU;

– скопировать данные из оперативной памяти в память видеокарты;

– проинициировать функции из загруженного модуля, указав степень распараллеливания;

– обработать данные;

– скопировать результаты из памяти GPU в память CPU.

# Оценка эффективности обработки данных

Из вышеперечисленных акций для работы на GPU оценим время обработки данных, как ключевого момента эффективности, и всех вспомогательных акций.

В случае сложения векторов ускорение обработки в зависимости от их размерности (от 8 до 256 элементов) на GPU по сравнению с CPU показано на рис. 3.

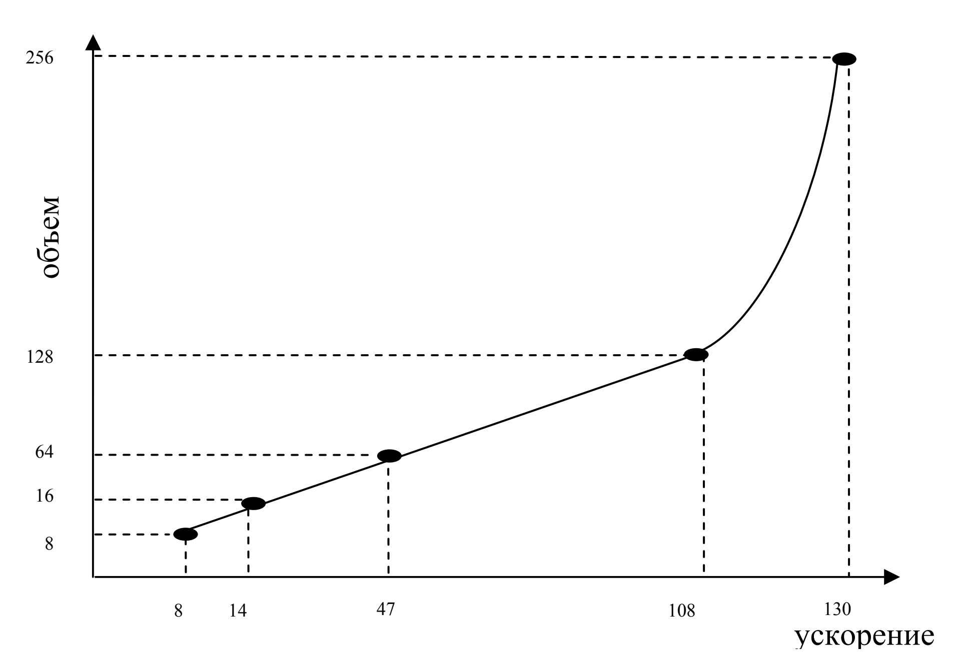



Рисунок 3 – Ускорение обработки

Эксперименты проводились на CPU Intel Core 2 Duo с тактовой частотой 1.8 GHz и GDU Nvidia GEFORCE 9600 GT с количеством потоковых процессоров 64 и объемом разделяемой памяти 16384 байт.

Что касается остальных этапов выполнения, то несравненно большее время занимает копирование данных из оперативной памяти CPU в память видеокарты и обратно, а также выделение памяти для решения поставленной задачи.

Второй эксперимент был проведен на примере умножения трех и двух матриц с последующим их сложением (ABC + DE). Соответствующее ускорение вычисления данной задачи для различных объемов исходных данных представлено в табл. 1. А удельный вес каждого этапа обработки при решении вышеизложенной задачи приведено в табл. 2.

Таблица 1 – Общее время выполнения программы

| Dimension of input data | GPU (ns) | CPU (ns) | Speed-up (times) |
|---|---|---|---|
| 128×128 | 538309478 | 19226187 | 0,035715862 |
| 256×256 | 494709803 | 129318391 | 0,261402524 |
| 512×512 | 484937052 | 1015339380 | 2,093755006 |
| 1024×1024 | 567429660 | 43473669698 | 76,61508159 |

Таблица 2 – Время выполнения каждого этапа программы

| | | |
|---|---|---|
| Memory allocation | 546718 | 1,62% |
| Copy data from host to device | 10042897 | 29,76% |
| Execution | 155327 | 0,46% |
| Copy data from device to host | 22239977 | 65,89% |
| Clean up | 766299 | 2,27% |

Из табл. 1 и 2 видно, что большую часть времени занимает операция копирования данных из памяти CPU к видеокарте и обратная запись результата. Время непосредственной обработки занимает доли процента.

Третий эксперимент проводился по обработке текста путем циклического сдвига на заданное количество бит при большом размере исходного массива. Он показывает, что 85 – 87% всего времени тратится на копирование в память процессоров.

## Требования к задачам для эффективного выполнения на GPU

Анализ технологии обработки задач на видеокартах, архитектурных решений CPU и GPU, а также проведение ряда экспериментов на таком тандеме позволяет утверждать, что в первую очередь следует предлагать задачи, которые хорошо распараллеливаются на сотни потоков. Особо значимое ускорение можно получить, если одни и те же инструкции применяются к огромным массивам данных.

Следующим требованием можно назвать отсутствие взаимодействий между обрабатываемыми потоками или «слабое» взаимодействие.

Среди остальных свойств задач (программ) для обработки на GPU можно назвать:

– минимальное количество сложных для обработки операций: деление, возведение в отрицательную степень и т.д.;

– отсутствие в алгоритмах множественного ветвления;

– небольшой объем данных, передаваемых к видеокарте и от нее в оперативную память CPU.

Безусловно, иногда указанные требования можно частично обойти за счет тщательного анализа алгоритма решения задачи и создания дополнительных средств, снижающих влияние указанных и других требований на время полного цикла решения задач. Но все это надо делать необычайно корректно.

## Заключение

Теоретические исследования и различные практические эксперименты показывают, что при тщательной подготовке алгоритмов и подборе соответствующих задач с большими параллельно обрабатываемыми фрагментами время вычислений сокращается в десятки и сотни раз. Безусловно, следует потратить время на изучение технологии, постановки задач на GPU, но потраченное время дает затем значительный эффект.

Особенно значимые результаты можно наблюдать при использовании тандема (CPU, GPU) при достойном планировании ресурсов.

## Литература

1. Буза М.К. Методы и средства распределенной обработки данных / М.К. Буза, Л.Ф. Зимянин // Выбраныя навуковыя працы Беларускага Дзяржаунага Універсітэту : у 7 т. – Мн.: БДУ, 2001. – Т. 6. Матэматыка. – С. 92-116.
2. Буза М.К. Системы параллельного действия / М.К. Буза. – Минск : БГУ, 2009. – 415 с.
3. NVIDIA CUDA – неграфические вычисления на графических процессорах // IXBT [Электронный ресурс]. – 2011. – Режим доступа : http://www.ixbt.com/video3/ cuda-l.shtml.

## Literatura

1. Buza M.K. VybranyjanavukovyjapracyBelaruskagaDzjarzhaunagaUniversitjetu : u 7 t. T 6. Matjematyka. Mn.: BDU. 2001. S. 92-116.
2. Buza M.K. Sistemyparallel’nogodejstvija. Minsk : BGU. 2009. 415 s.
3. NVIDIA CUDA – negraficheskievychislenijanagraficheskihprocessorah. IXBT. 2011. http://www.ixbt.com/video3/ cuda-l.shtml.

*М.К. Буза*

**Паралельні обчислення на графічних процесорах**

Розглядається один із підходів скорочення часу обчислень добре розпаралелювальних алгоритмів, заснований на сумісним використанні класичних та графічних процесорів. Наводиться аналіз ефективності такої організації обчислень та з’ясовуються «вузькі» місця. На конкретних прикладах показуються можливості подібного тандема та формулюються вимоги до алгоритмів для їх ефектного використання.

*Статья поступила в редакцию 04.07.2011.*